## Problem Definition

In this challenge, we need to identify which customers will make a specific transaction in

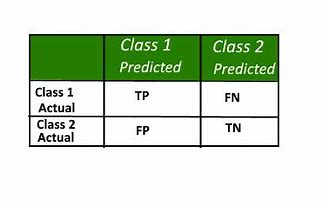
the future, irrespective of the amount of money transacted.

## AIM

The task is to predict the value of target column in the test set

## Evaluation basis

Confusion Matrix



Accuary:correct predictions / total predictions \* 100

precision:true positive/(true positive+false positive)

Recall:true positive/(true positive+false negative)

Roc\_AUC:AUC stands for "Area under the ROC Curve” and ROC curve is plotting between false positive rate and true positive rate where true positive in y-axis and false positive on x-axis. it is very good metric get an idea how good a metrics

is.it's value varies from 0.5 to 1.

## How to run and deploy code ?

I am sharing jupyter notebook for python code and rscript(R code)

I would suggest only run model naive bayes and lightgbm As they are giving more accurate result and it will take lot less time to run campared to other model.I will mention the output of most of model in this report

## Problem Feature

1. train.csv - the training set.

2. test.csv - the test set.

## Import libraries

### IN Python

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import StratifiedKFold

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

import pandas as pd

import numpy as np

import os

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score,roc\_auc\_score,roc\_curve,auc,recall\_score,precision\_score

from sklearn.metrics import accuracy\_score,roc\_auc\_score,roc\_curve,auc,recall\_score,precision\_score

import eli5

from sklearn.feature\_selection import SelectFromModel

from eli5.sklearn import PermutationImportance

import imblearn

from imblearn.over\_sampling import SMOTE

from sklearn.feature\_selection import SelectFromModel

import lightgbm as lgb

### IN R

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information",

"MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')

require("pROC")

lapply(x, require, character.only = TRUE)

rm(x)

## Importing files

### IN python

train=pd.read\_csv("train.csv")

test=pd.read\_csv("test.csv")

### IN R

train=read.csv("train.csv")

test=read.csv("test.csv")

## Exploratory Data Analysis(EDA)

### files Overview

#### IN python

train.shape()

test.shape()

print(train.info())

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 200000 entries, 0 to 199999

Columns: 202 entries, ID\_code to var\_199

dtypes: float64(200), int64(1), object(1)

memory usage: 308.2+ MB

None

print(test.info())

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 200000 entries, 0 to 199999

Columns: 201 entries, ID\_code to var\_199

dtypes: float64(200), object(1)

memory usage: 306.7+ MB

None

#### IN R

head(train,10)

str(train)

str(test)

As we can see that there are 200 numerical variable,1 target numerical variable,1 categorical variable with 2 lakh rows of data and

categorical variable is useless as it meaning the index So we can remove it

### distribution of target variable in train variable

#### IN python

train['target'].value\_counts()

0 179902

1 20098

Name: target, dtype: int64

#### IN R

table(train$target)

So we can see that there is class imbalance problem

### Missing value analysis

#### IN Python

train.isnull().any().value\_counts()

False 202

dtype: int64

#As Count equal to no of column So there are no missing value

#### In R

missing\_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))

So there are No missing value

### Distribution of some feature

#### IN python

print('distribution of first 25 variable');

plt.figure(figsize=(25,25))

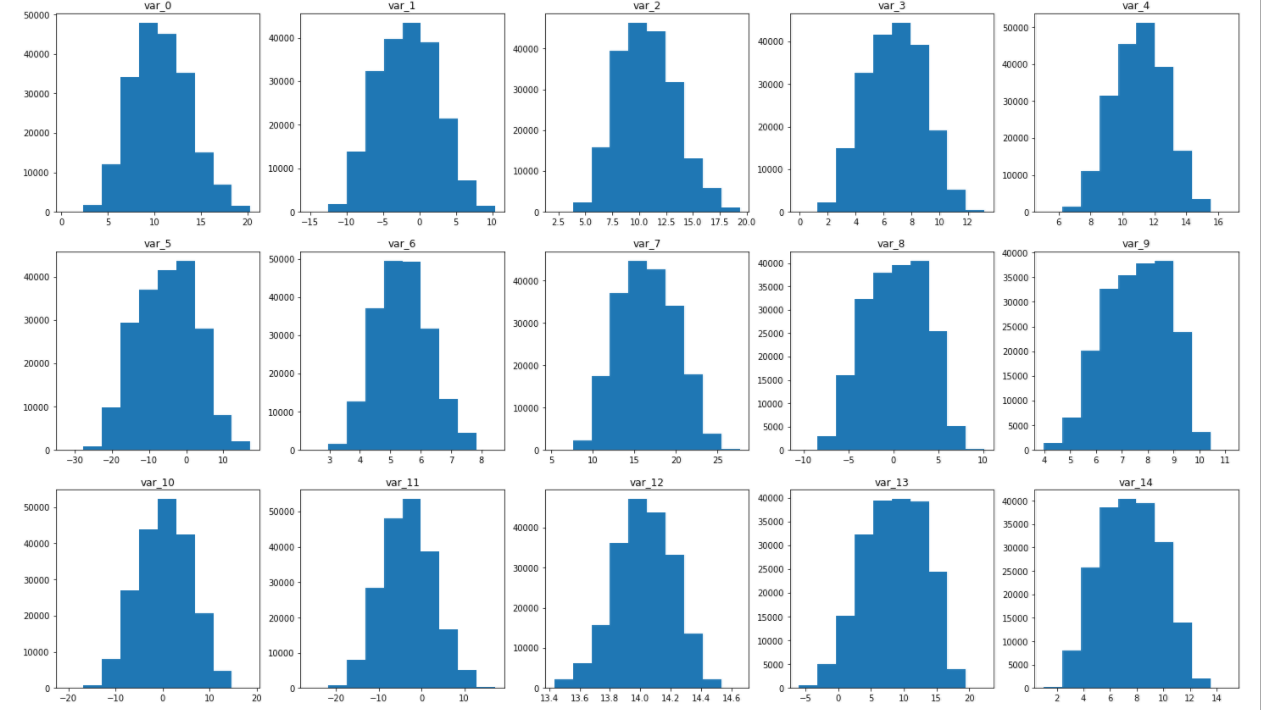
for i,col in enumerate(list(train.columns)[2:27]):

plt.subplot(5,5,i+1)

plt.hist(train[col])

plt.title(col)

So most of the variable are normally distributed with some skewness



#### IN R

range<-c(1:9)

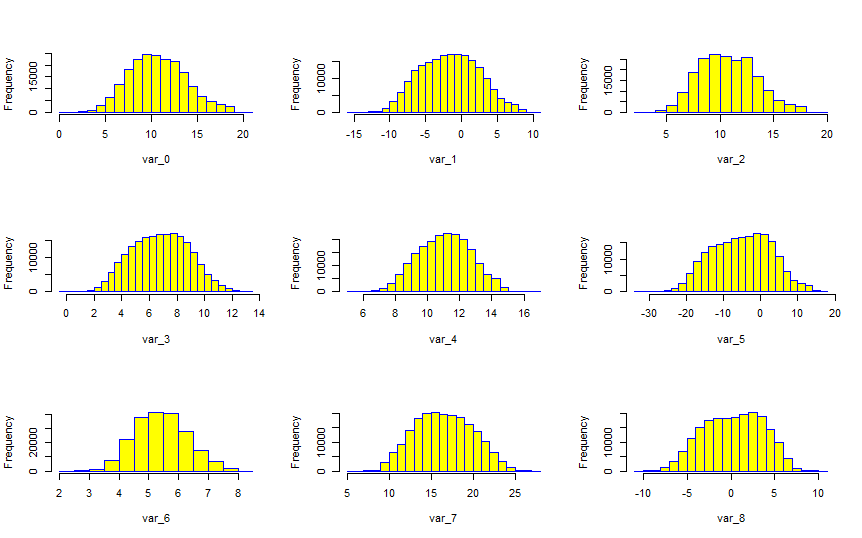
grid<-matrix(c(1:9),nrow=3,ncol = 3,byrow = TRUE)

layout(grid)

for(i in range){

hist(train[,i+2],col = "yellow",border = "blue",xlab =colnames(train[i+2]),main = "")

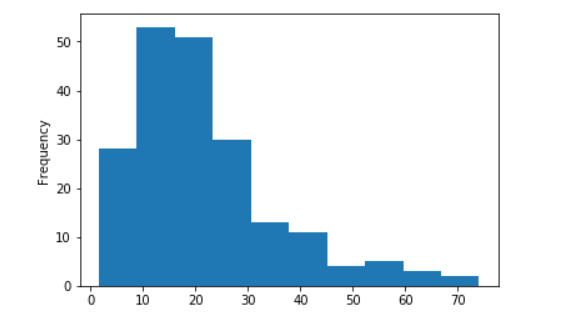
}



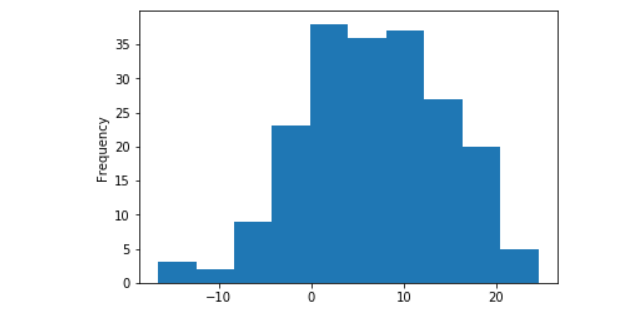
### Distribution of mean,median,std,min,max

#### IN Python

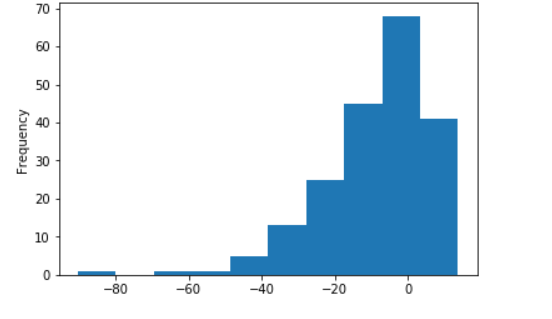
train[train.columns[2:]].max().plot('hist');



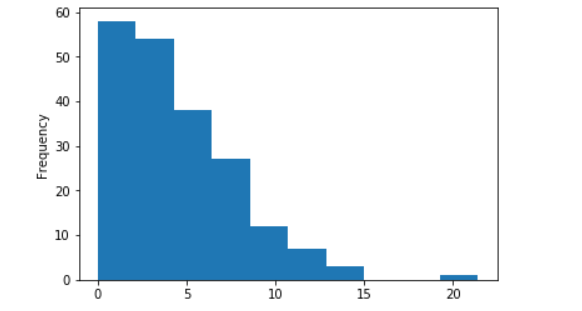
train[train.columns[2:]].mean().plot('hist');



train[train.columns[2:]].min().plot('hist');



train[train.columns[2:]].std().plot('hist');



#### IN R

grid<-matrix(c(1:4),nrow=2,ncol = 2,byrow = TRUE)

layout(grid)

variable\_mean=c()

variable\_min=c()

variable\_max=c()

for(i in c(1:200))

{

variable\_mean<-c(mean(train[,i+2]),variable\_mean)

variable\_max<-c(max(train[,i+2]),variable\_max)

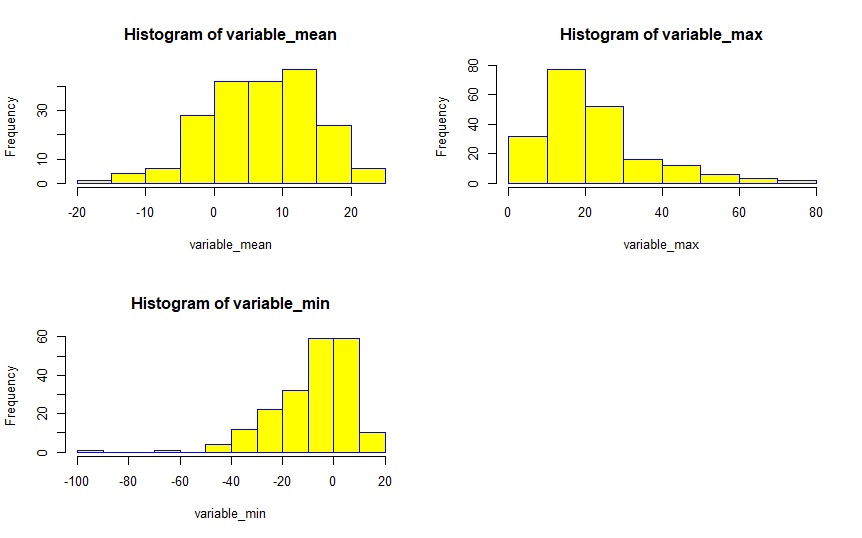
variable\_min<-c(min(train[,i+2]),variable\_min)

}

hist(variable\_mean,col = "yellow",border = "blue",main=)

hist(variable\_max,col = "yellow",border = "blue",main=)

hist(variable\_min,col = "yellow",border = "blue",main=)



### Correlation analysis

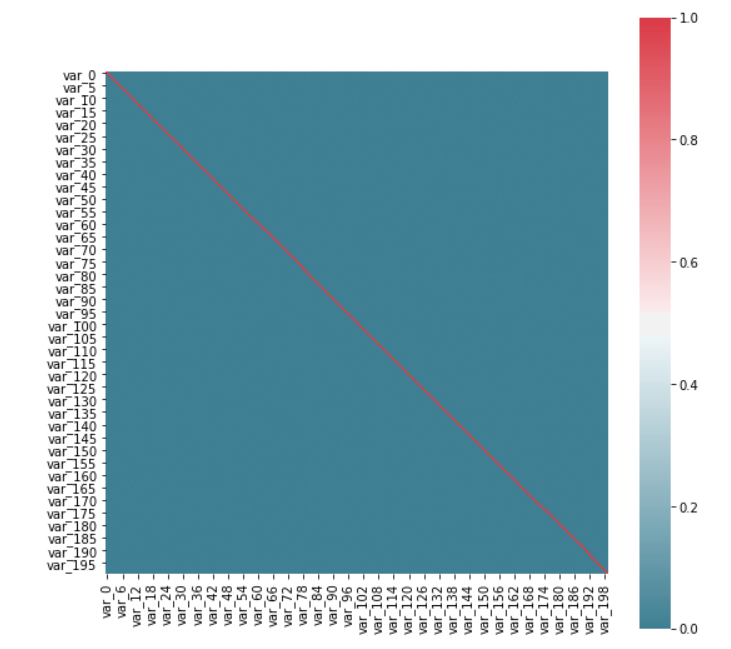
#### IN python

corr=train[train.columns[2:]].corr().abs()

f, ax = plt.subplots(figsize=(9, 9))

#Plot using seaborn library

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=True), square=True, ax=ax)



### IN R

corrgram(train[,3:202], order = F,

upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

### Checking the distribution of target variable value over few features

#### In Python

feat1, feat2 = 'var\_12', 'var\_6'

fig = plt.subplots(figsize=(15, 5))

#plot pdf feature 1

plt.subplot(1, 2, 1)

sns.kdeplot(train[feat1][train['target'] == 0], color="b", label = 'target = 0')

sns.kdeplot(train[feat1][train['target'] == 1], color="r", label = 'target = 1')

plt.title(feat1)

plt.xlabel('Feature Values')

plt.ylabel('Probability')

#plot pdf feat 2

plt.subplot(1,2, 2)

sns.kdeplot(train[feat2][train['target'] == 0], color="b", label = 'target = 0\_train')

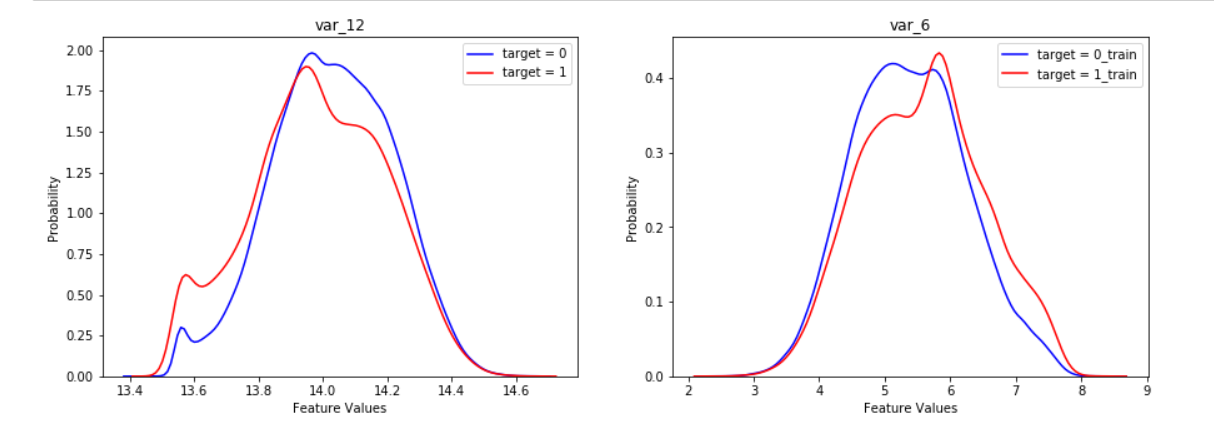
sns.kdeplot(train[feat2][train['target'] == 1], color="r", label = 'target = 1\_train')

plt.title(feat2)

plt.xlabel('Feature Values')

plt.ylabel('Probability')

plt.show()



IN R

As we can see that their is no collinearity between any feature and their is need for outlinear anlysis(only 50 to rows are going to get remove which negligible) and As i am planning to use decision tree,and random forest for now So there is no need of feature scaling for now. In naive bayes I will use scale function to normalizes the value

## Model Development

Partition the train into training data and validation data in 80/20 and removing removing IN\_code column

#### In python

X = train.drop(['ID\_code', 'target'], axis=1)

Y = train['target']

train\_X, val\_X, train\_y, val\_y = train\_test\_split(X, Y, test\_size=.2,stratify=y)

def output\_score(Y,Predictions):

print("area under curve in ROC: {:<8.5f}".format(roc\_auc\_score(Y, Predictions)))

print("accuracy\_score: {:<8.5f}".format(accuracy\_score(Y,Predictions)))

print("recall\_score: {:<8.5f}".format(recall\_score(Y,Predictions)))

print("precision\_score : {:<8.5f}".format(precision\_score(Y, Predictions)))

#### IN R

set.seed(1234)

train.index = createDataPartition(train$target, p = .80, list = FALSE)

train\_file=train[train.index,]

train\_file=train\_file[,-1]

val\_file=train[-train.index,]

val\_file=val\_file[,-1]

train\_file$target<-as.factor(train\_file$target)

val\_file$target<-as.factor(val\_file$target)

### Decision tree:

First Model we say trying is decision As this is a classification problem and decision tree is one of most basic classification model.

#### In python

C50\_model = DecisionTreeClassifier(criterion='entropy').fit(train\_X,train\_y)

predict\_Tree=C50\_model.predict(val\_X)

Tree=pd.DataFrame(predict\_Tree)

Triee[0].value\_counts()

0 35833

1 4167

Name: 0, dtype: int64

val\_y.value\_counts()

0 35980

1 4020

Name: target, dtype: int64

CM=pd.crosstab(val\_y,predict\_Tree)

CM

output\_score(val\_y,predict\_Tree)

area under curve in ROC: 0.55188

accuracy\_score: 0.83503

recall\_score: 0.19751

precision\_score : 0.19054

#### IN R

C50\_model = C5.0(target ~.,train\_file, rules = TRUE)

C50\_Predictions = predict(C50\_model,val\_file[-1], type = "class")

C50\_Predictions

confmatrix=table(val\_file$target,C50\_Predictions)

ConfMatrix=as.matrix(confmatrix)

TN=ConfMatrix[1]

FN=ConfMatrix[2]

FP=ConfMatrix[3]

TP=ConfMatrix[4]

accuary=(TN+TP)/(TN+TP+FN+FP)

accuary

#0.89

precision <- TP/(TP+FP)

precision

#0.38

recall <- TP/(TP+FN)

recall

#0.07

roc\_obj <- roc(as.numeric(val\_file[,1]), as.numeric(C50\_Predictions))

auc(roc\_obj)

#0.54

### Randomforest:

#### In python

rfc\_model = RandomForestClassifier(n\_estimators=10).fit(train\_X, train\_y)

RF\_Predictions = rfc\_model.predict(val\_X)

val\_y.value\_counts()

0 35980

1 4020

Name: target, dtype: int64

x=pd.DataFrame(RF\_Predictions)

x[0].value\_counts()

0 39878

1 122

Name: 0, dtype: int64

CM = pd.crosstab(val\_y, RF\_Predictions)

CM #confusuion matrix

output\_score(val\_y, RF\_Predictions)

area under curve in ROC: 0.50881

accuracy\_score: 0.90025

recall\_score: 0.01891

precision\_score : 0.62295

#### IN R

RF\_model = randomForest(target ~ ., train\_file, importance = TRUE, ntree =25)

RF\_Predictions = predict(RF\_model,val\_file[-1] )

ConfMatrix\_RF = table(val\_file[,1], RF\_Predictions)

ConfMatrix\_RF=as.matrix(ConfMatrix\_RF)

TN=ConfMatrix\_RF[1]

FN=ConfMatrix\_RF[2]

FP=ConfMatrix\_RF[3]

TP=ConfMatrix\_RF[4]

accuary=(TN+TP)/(TN+TP+FN+FP)

accuary

precision <- TP/(TP+FP)

precision

recall <- TP/(TP+FN)

recall

roc\_obj <- roc(as.numeric(as.numeric(val\_file[,1])), as.numeric(RF\_Predictions))

roc\_obj

roc\_obj <- roc(as.numeric(val\_file[,1]), as.numeric(RF\_Predictions))

auc(roc\_obj)

We have tries these two model both of them are giving horrible recall score,precision score

As we have seen in EDA there is a class inbalance problem We can solve them it by applying SMOTE

SMOTE:Synthetic Minority Over-sampling Technique :In this method generate new minority instances between existing (real) minority instances.

#### IN python

def smote\_value\_update(train\_X,train\_y):

smt=SMOTE()

new\_train\_X,new\_train\_y=smt.fit\_sample(train\_X,train\_y)

return new\_train\_X,new\_train\_y

#### IN R

tab<-as.data.frame(table(train\_file$target))

newtrain\_file <- SMOTE(target ~ ., train\_file,prec.over=tab[1,2]-tab[2,2])

#### IN python(Random forest with SMOTE)

smt=SMOTE()

new\_train\_X,new\_train\_y=smt.fit\_sample(train\_X,train\_y)

rfc\_model = RandomForestClassifier(n\_estimators=10).fit(new\_train\_X,new\_train\_y)

RF\_Predictions = rfc\_model.predict(val\_X)

x=pd.DataFrame(RF\_Predictions)

x[0].value\_counts()

0 37280

1 2720

Name: 0, dtype: int64

val\_y.value\_counts()

0 35980

1 4020

Name: target, dtype: int64

CM = pd.crosstab(val\_y, RF\_Predictions)

roc\_value = roc\_auc\_score(val\_y,rfc\_predict)

auc=accuracy\_score(val\_y,rfc\_predict)

recall=recall\_score(val\_y,rfc\_predict)

precision=precision\_score(val\_y,rfc\_predict)

print(roc\_value,auc,recall,precision)

0.5342423513339362 0.85755 0.12960199004975123 0.19154411764705884

#### IN R(Random forest with SMOTE)

tab<-as.data.frame(table(train\_file$target))

newtrain\_file <- SMOTE(target ~ ., train\_file,prec.over=tab[1,2]-tab[2,2])

RF\_model = randomForest(target ~ ., newtrain\_file, importance = TRUE, ntree =12)

RF\_Predictions = predict(RF\_model,val\_file[-1] )

ConfMatrix\_RF = table(val\_file[,1], RF\_Predictions)

ConfMatrix\_RF

ConfMatrix\_RF=as.matrix(ConfMatrix\_RF)

TN=ConfMatrix\_RF[1]

FN=ConfMatrix\_RF[2]

FP=ConfMatrix\_RF[3]

TP=ConfMatrix\_RF[4]

accuary=(TN+TP)/(TN+TP+FN+FP)

accuary

precision <- TP/(TP+FP)

precision

recall <- TP/(TP+FN)

recall

roc\_obj <- roc(as.numeric(as.numeric(val\_file[,1])), as.numeric(RF\_Predictions))

roc\_obj

### feature importance

There is a improvement in recall metric in expense of accuracy,precision value

let's look at feature importance

for python I will use Permutation Importance for computing the features importance

For R I will use inbuilt function only(As I could't find Permutation Importance in R)

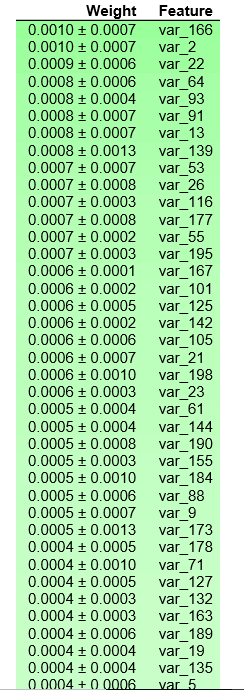
First what is permutation importance

Permutation Importance. eli5 provides us a way to compute feature importances for any estimator by measuring how score decreases when a feature is not available

#### In python

perm=PermutationImportance(rfc\_model).fit(val\_X,val\_y)

eli5.show\_weights(perm,feature\_names=val\_X.columns.tolist(),top=150)



#### IN R(Random Forest)

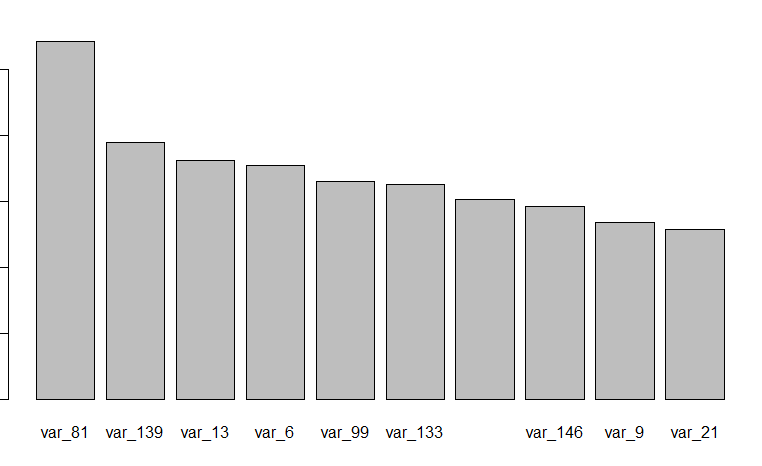
gbmImp <- varImp(RF\_model, scale = FALSE)

gbmImp<-gbmImp[order(-gbmImp$`0`),]

gbmImp

c<-rownames(gbmImp[1:10,])

barplot(gbmImp[1:10,1],names.arg=c)



### Hypertuning

let's try to tune the parameter but I have applied gridseasrch on parameter and it is taking a lot of time(more than 20 hours but still computing both in R and python) So i will try to optimize the parameter by visualization.

#### IN Python

No of estimators

n\_estimators = [1, 2, 4, 8, 16]

train\_results = []

test\_results = []

for estimator in n\_estimators:

rf = RandomForestClassifier(n\_estimators=estimator, n\_jobs=-1)

rf=rf.fit(new\_train\_X, new\_train\_y)

train\_pred = rf.predict(new\_train\_X)

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(new\_train\_y, train\_pred)

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

train\_results.append(roc\_auc)

y\_pred = rf.predict(val\_X)

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(val\_y, y\_pred)

roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

test\_results.append(roc\_auc)

lines = plt.plot(n\_estimators, train\_results,n\_estimators,test\_results)

plt.setp(lines[0], linewidth=4)

plt.setp(lines[1], linewidth=2)

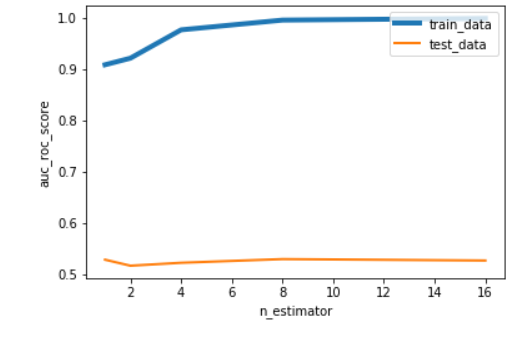
plt.legend(('train\_data','test\_data'),loc='upper right')

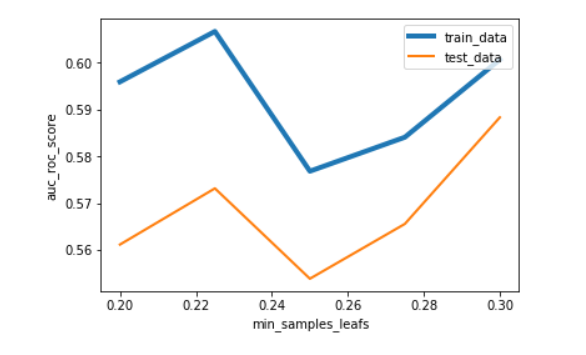
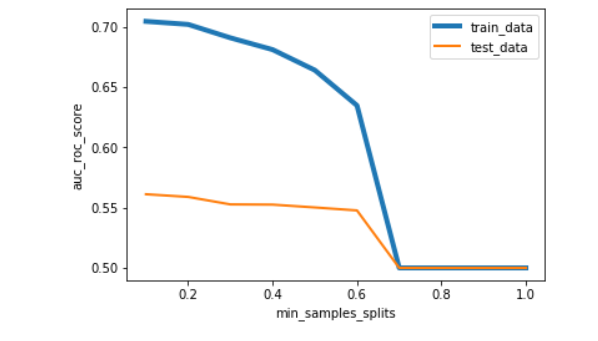
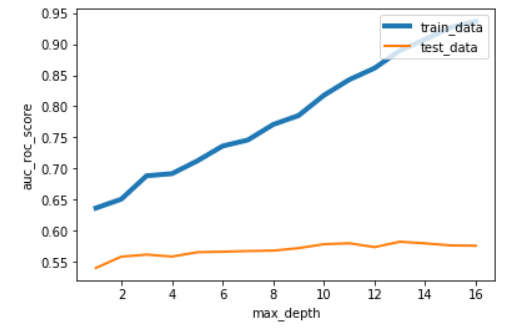
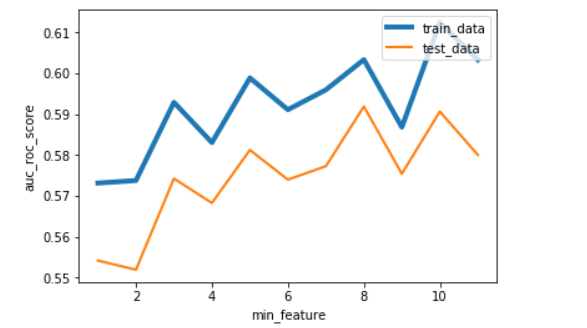
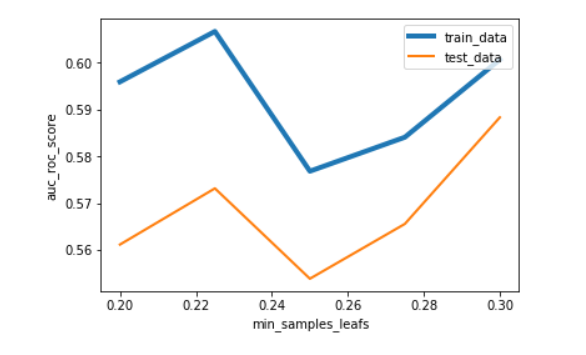
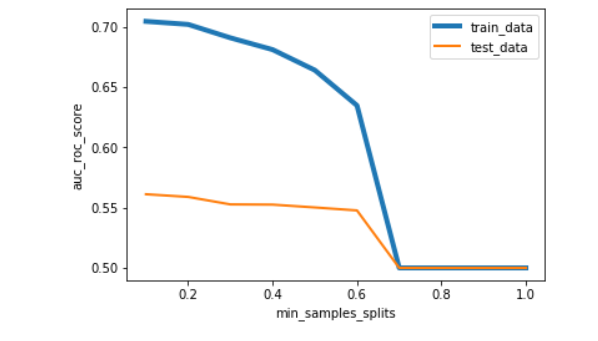
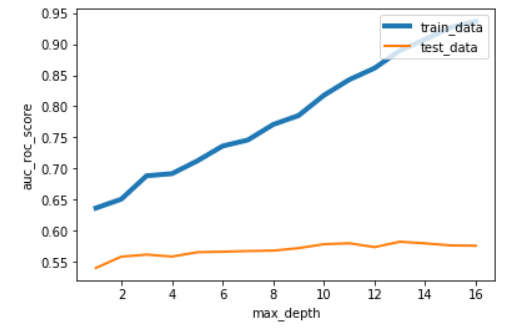
plt.xlabel("n\_estimator")

plt.ylabel("auc\_roc\_score")

plt.show()

In the same way we will optimize other parameter like max\_depth,min\_samples\_splits,min\_feature





#### 

#### 

#### 

#### IN R(random Forest)

Range=c(1,2,4,8,16,32)

train\_results<-c()

test\_results<-c()

for (i in Range)

{

RF\_model = randomForest(target ~ ., newtrain\_file, importance = TRUE, ntree =i)

train\_pred = predict(RF\_model,newtrain\_file[,-1])

roc\_obj <- roc(as.numeric(newtrain\_file[,1]), as.numeric(train\_pred))

train\_results<-c(train\_results,auc(roc\_obj))

train\_pred = predict(RF\_model,val\_file[,-1])

roc\_obj <- roc(as.numeric(as.numeric(val\_file[,1])), as.numeric(train\_pred))

test\_results<-c(test\_results,auc(roc\_obj))

}

train\_results\_dataFrame=as.data.frame(list(train\_results,Range))

test\_results\_dataFrame=as.data.frame(list(test\_results,Range))

cols = c("AU", "value")

colnames(train\_results\_dataFrame) = cols

colnames(test\_results\_dataFrame) = cols

p = ggplot() +

geom\_line(data = train\_results\_dataFrame, aes(x = value, y = AU), color = "blue") +

geom\_line(data = test\_results\_dataFrame, aes(x = value, y = AU), color = "red") +

xlab('no of tree') +

ylab('AUC Score')

print(p)

In the same way we will optimize other parameter in same way

### model development with optimized parameter

#### IN python(random forest with optimized parameter)

rfc\_model=RandomForestClassifier(n\_estimators=32,max\_depth=7,min\_samples\_split=0.1,min\_samples\_leaf=0.28,max\_features=7).fit(new\_train\_X,new\_train\_y)

rfc\_predict=rfc\_model.predict(val\_X)

rand\_predict= pd.DataFrame(rfc\_predict)

rand\_predict[0].value\_counts()

0 24958

1 15042

Name: 0, dtype: int64

CM=pd.crosstab(val\_y,rfc\_predict)

CM

output\_score(val\_y,rfc\_predict)

area under curve in ROC: 0.61937

accuracy\_score: 0.64220

recall\_score: 0.59080

precision\_score : 0.15789

there is increment is recall ,roc score but there is quite a decrement in accuracy

Then applying 10 fold cross validation

def random\_forest\_function(X\_fit, y\_fit, X\_val, y\_val):

X\_fit, y\_fit=smote\_value\_update(X\_fit, y\_fit)

rfc= RandomForestClassifier(n\_estimators=32,max\_depth=7,min\_samples\_split=0.1,min\_samples\_leaf=0.28,max\_features=7)

model=rfc.fit(X\_fit,y\_fit)

rfc\_predict = model.predict(X\_val)

rand\_predict= pd.DataFrame(rfc\_predict)

print(rand\_predict[0].value\_counts())

return roc\_auc\_score(y\_val,rfc\_predict)

y\_df = np.array(Y)

df\_ids=np.array(X.index)

skf = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=42)

skf.get\_n\_splits(df\_ids, y\_df)

t=0

for counter, ids in enumerate(skf.split(df\_ids, y\_df)):

print('\nFold {}'.format(counter+1))

X\_fit, y\_fit = X.values[ids[0]], y\_df[ids[0]]

X\_val, y\_val = X.values[ids[1]], y\_df[ids[1]]

auc\_ro=random\_forest\_function(X\_fit, y\_fit, X\_val, y\_val)

t=t+auc\_ro

print(auc\_ro)

t/10

output:-

Fold 1

0 12181

1 7820

Name: 0, dtype: int64

0.6120376523253335

Fold 2

0 12207

1 7794

Name: 0, dtype: int64

0.605016894848751

Fold 3

0 12212

1 7788

Name: 0, dtype: int64

0.6181712338806247

Fold 4

0 12242

1 7758

Name: 0, dtype: int64

0.595221778821291

Fold 5

0 12188

1 7812

Name: 0, dtype: int64

0.5970395382730593

Fold 6

0 12050

1 7950

Name: 0, dtype: int64

0.6086908426184807

Fold 7

0 12393

1 7607

Name: 0, dtype: int64

0.6052260929925137

Fold 8

0 12340

1 7660

Name: 0, dtype: int64

0.6228349635922665

Fold 9

0 12129

1 7870

Name: 0, dtype: int64

0.5914180932883735

Fold 10

0 12245

1 7754

Name: 0, dtype: int64

0.6112425021256486

0.6066899592766342

#### IN R

Hypertuning randomforest

RF\_model = randomForest(target ~ ., newtrain\_file, importance = TRUE, ntree=32,mtry=7,nodesize=0.28)

RF\_Predictions = predict(RF\_model,val\_file[-1] )

Conf\_matrix = table(val\_file[,1], RF\_Predictions)

Conf\_matrix=as.matrix(Conf\_matrix)

TN=Conf\_matrix[1]

FN=Conf\_matrix[2]

FP=Conf\_matrix[3]

TP=Conf\_matrix[4]

accuary=(TN+TP)/(TN+TP+FN+FP)

accuary

# 0.857075

precision <- TP/(TP+FP)

precision

# 0.2356467

recall <- TP/(TP+FN)

recall

#0.1848552

roc\_obj <- roc(as.numeric(val\_file[,1]), as.numeric(RF\_Predictions))

auc(roc\_obj)

#Area under the curve: 0.5587

So there is not much Improvement even after hypertuning

IN R(for cross validation I have not used any libraries I just have cut data set into 5 part and applied cross validation with optimized parameter

#k fold cross validition in R

#Create 5 equally size folds

train1<-train[sample(nrow(train)),]

train1$target<-as.factor(train1$target)

#Create 10 equally size folds

folds <- cut(seq(1,nrow(train1)),breaks=5,labels=FALSE)

k\_acc<-c()

#Perform 5 fold cross validation

for(i in 1:5){

#Segement your data by fold using the which() function

testIndexes <- which(folds==i)

valData <- train\_file[testIndexes, ]

trainData <- train\_file[-testIndexes, ]

tab<-as.data.frame(table(trainData$target))

newtrain\_file1 <- SMOTE(target ~ ., trainData ,prec.over=tab[1,2]-tab[2,2])

RF\_model = randomForest(target ~ ., newtrain\_file1, importance = TRUE, ntree =12,mtry=8,nodesize=.28)

train\_pred = predict(RF\_model,valData[,-1])

roc\_obj <- roc(as.numeric(as.numeric(valData[,1])), as.numeric(train\_pred))

auc(roc\_obj)

K\_acc<-c(auc(roc\_obj),k\_acc)

}

sum(k\_acc)/5

#0.554

### Naive Bayes

So even with optimizing parameter we are getting very poor score in all metrics

AS all features have no correlation We could use naive bayes we just have to normalizes all the features

#### IN python

from sklearn import preprocessing

from sklearn.naive\_bayes import GaussianNB

standardized\_X = preprocessing.scale(X)

train\_XNB, val\_XNB, train\_yNB, val\_yNB = train\_test\_split(standardized\_X, Y,test\_size=.2,stratify=Y)

#Naive Bayes implementation

NB\_model = GaussianNB().fit(train\_XNB,train\_yNB)

NB\_Predictions = NB\_model.predict(val\_XNB)

output\_score(val\_yNB,NB\_Predictions)

area under curve in ROC: 0.66846

accuracy\_score: 0.92053

recall\_score: 0.35299

precision\_score : 0.71057

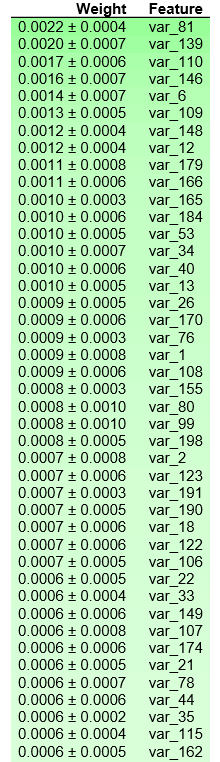
we are getting score much better then both decision tree and random forest

we can say this model better then rest

Now let's look at feature importance

perm=PermutationImportance(NB\_model).fit(val\_XNB,val\_yNB)

eli5.show\_weights(perm,feature\_names=X.columns.tolist(),top=150)



#### IN R

scale\_train=as.data.frame(scale(train[,3:202]))

scale\_train$target=as.factor(train$target)

scale\_train\_file=scale\_train[train.index,]

scale\_val\_file=scale\_train[-train.index,]

#Develop model

NB\_model = naiveBayes(target ~ ., data = scale\_train\_file)

NB\_Predictions = predict(NB\_model, scale\_val\_file[,-201], type = 'class')

Conf\_matrix = table(observed = scale\_val\_file[,'target'], predicted = NB\_Predictions)

Conf\_matrix=as.matrix(Conf\_matrix)

TN=Conf\_matrix[1]

FN=Conf\_matrix[2]

FP=Conf\_matrix[3]

TP=Conf\_matrix[4]

accuary=(TN+TP)/(TN+TP+FN+FP)

accuary

#0.921

precision <- TP/(TP+FP)

precision

#0.72

recall <- TP/(TP+FN)

recall

#0.36

roc(as.numeric(scale\_val\_file[,201]), as.numeric(NB\_Predictions))

#0.674

imp<-c()

for(i in c(1:200))

{

imp=c(permutationImportance(scale\_val\_file,colnames(scale\_val\_file[i]), 'target', NB\_model),imp)

}

imp=as.dataFrame(list(imp,colnames(scale\_val\_file)))

colnames(imp)=c(1:2)

imp=imp[order(imp$'1', decreasing=TRUE), ]

barplot(imp[1:10,1],names.arg=imp[1:10,2])

# it taking very long time so could't able provide the features importance

Coming to last method I am going to use is one of famous booting method LightGBM but before using this model brief description about it.

## Light *GBM*

In recant years there are some algo which performing were well with large Dataset(provide very fast implementation and give very  great result) these algo are Light *GBM*, CatBoost, XGBoost .For this project I have decided to use   LightGBM

Model for classification.

### Brief Working principle of Light *GBM*

 Light *GBM* is a gradient boosting framework that uses tree based learning algorithm. The advantage of light GBM:

·        Faster training speed and higher efficiency.

·        Lower memory usage.

·        Better accuracy.

·        Support of parallel and GPU learning.

·        Capable of handling large-scale data.

#### Now Question is what is gradient Gradient Boosting Framework ?

Boosting is an ensemble technique in which the predictors are not made independently, but sequentially . Main idea behind this that next learner/predictor will learn from the mistake of previous predictors.So the probability of a observation apperaing  in next model depending  upon the error(Wrong Classification for classification model).those observation that have been  wrongly classify have a higher chance of getting selected for next prediction

#### Gradient Boosting algorithm

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. (Wikipedia definition).

In Supervise learning we have to define and minimize the loss function. Let consider an example

Say we have mean squared error (MSE) as loss  as:

We want our predictions, such that our loss function (MSE) is minimum. By using **gradient descent** and updating our predictions based on a learning rate, we can find the values where MSE is minimum.

So, the reasoning behind gradient boosting algorithm is to repetitively try to use the patterns in residuals and strengthen a model with weak predictions and make it better.when we reached the point there is no change in residual pattern when we should stop the model(if not it will lead to overfitting).our main aim to minimize the lose function so that loss function of test data should reach minima.

#### How tree based learning algorithm used by Light GBM  **and why it is very popular** ?

The learning algo of light gbm is different from other tree method.light gbm grows vertically while other tree model grow horizontly.So basically there is a leaf wise growth compared to other tree level wise growth. Light GBM is prefixed as ‘Light’ because of its high speed. Light GBM can handle the large size of data and takes lower memory to run**.** Another reason of why Light GBM is popular is because it focuses on accuracy of results.

### Light GBM  Parameters:

##### Control Parameter

max\_depth: It describes the maximum depth of tree. This parameter is used to handle model overfitting. As higher depth can result in overfitting and if You feel that your model is overfitting try to reduce the depth of tree.

min\_data\_in\_leaf: As the name suggest it is minimum number of the records that a leaf may have. Default value is 20.

feature\_fraction:  feature fraction in Light *GBM* percentage of   parameters randomly to be selected in each time for building trees. Max value is 1 that mean’s all feature are selected every iteration.

**bagging\_fraction**: specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting. Just like feature\_fraction but this will randomly select part of data without resampling

early\_stopping\_round: This parameter can help you speed up your analysis. Model will stop training if one metric of one validation data doesn’t improve.

Boosting:efines the type of algorithm you want to run, default=gdbt

num\_boost\_round: Number of boosting iterations, typically 100+

metric: again one of the important parameter as it specifies loss for model building

#### IN python

param = {

'bagging\_freq': 5,

'bagging\_fraction': 0.4,

'boost\_from\_average':'false',

'boost': 'gbdt',

'feature\_fraction': 1,#0.05

'learning\_rate': 0.01,

'max\_depth': -1,

'metric':'auc',

'min\_data\_in\_leaf': 80,

'min\_sum\_hessian\_in\_leaf': 10.0,

'num\_leaves': 13,

'num\_threads': 8,

'tree\_learner': 'serial',

'objective': 'binary',

'verbosity': 1

}

features=train\_X.columns

#we need convert features in lgb in built Dataset

trn\_data = lgb.Dataset(train\_X, label=train\_y)

val\_data = lgb.Dataset(val\_X, label=val\_y)

num\_round = 1000000

clf = lgb.train(param, trn\_data, num\_round, valid\_sets = [trn\_data, val\_data], verbose\_eval=1000, early\_stopping\_rounds = 3000)

oof = clf.predict(val\_X, num\_iteration=clf.best\_iteration)

print("roc\_auc\_score score: {:<8.5f}".format(roc\_auc\_score(val\_y, oof)))

roc\_auc\_score score: 0.89677

##### with 10 fold cross validation

oof = np.zeros(len(X))

feature\_importance\_df = pd.DataFrame()

folds = StratifiedKFold(n\_splits=10, shuffle=False, random\_state=44000)

for fold\_, (trn\_idx, val\_idx) in enumerate(folds.split(X.values, Y.values)):

print("Fold {}".format(fold\_))

print(val\_idx)

print(trn\_idx)

trn\_data = lgb.Dataset(X.iloc[trn\_idx][features], label=Y.iloc[trn\_idx])

val\_data = lgb.Dataset(X.iloc[val\_idx][features], label=Y.iloc[val\_idx])

num\_round = 1000000

clf = lgb.train(param, trn\_data, num\_round, valid\_sets = [trn\_data, val\_data], verbose\_eval=1000, early\_stopping\_rounds = 3000)

oof[val\_idx] = clf.predict(X.iloc[val\_idx][features], num\_iteration=clf.best\_iteration)

fold\_importance\_df = pd.DataFrame()

fold\_importance\_df["Feature"] = features

fold\_importance\_df["importance"] = clf.feature\_importance()

fold\_importance\_df["fold"] = fold\_ + 1

feature\_importance\_df = pd.concat([feature\_importance\_df, fold\_importance\_df], axis=0)

print("CV score: {:<8.5f}".format(roc\_auc\_score(Y, oof)))

Output

Fold 0

[ 0 1 2 ... 20470 20474 20481]

[ 19956 19957 19958 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.887062 valid\_1's auc: 0.861069

[2000] training's auc: 0.914652 valid\_1's auc: 0.88395

[3000] training's auc: 0.927194 valid\_1's auc: 0.891992

[4000] training's auc: 0.935055 valid\_1's auc: 0.895212

[5000] training's auc: 0.94134 valid\_1's auc: 0.896834

[6000] training's auc: 0.947249 valid\_1's auc: 0.896983

[7000] training's auc: 0.952601 valid\_1's auc: 0.897567

[8000] training's auc: 0.957695 valid\_1's auc: 0.897479

[9000] training's auc: 0.962642 valid\_1's auc: 0.897073

[10000] training's auc: 0.967031 valid\_1's auc: 0.897018

Early stopping, best iteration is:

[7414] training's auc: 0.954782 valid\_1's auc: 0.897774

Fold 1

[19956 19957 19958 ... 40689 40696 40717]

[ 0 1 2 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.886931 valid\_1's auc: 0.862175

[2000] training's auc: 0.914824 valid\_1's auc: 0.884463

[3000] training's auc: 0.927012 valid\_1's auc: 0.891946

[4000] training's auc: 0.935008 valid\_1's auc: 0.895584

[5000] training's auc: 0.941389 valid\_1's auc: 0.896771

[6000] training's auc: 0.947098 valid\_1's auc: 0.897096

[7000] training's auc: 0.952582 valid\_1's auc: 0.897063

[8000] training's auc: 0.957612 valid\_1's auc: 0.896827

[9000] training's auc: 0.962494 valid\_1's auc: 0.89704

Early stopping, best iteration is:

[6340] training's auc: 0.949018 valid\_1's auc: 0.89716

Fold 2

[39914 39915 39916 ... 60569 60572 60581]

[ 0 1 2 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.887426 valid\_1's auc: 0.857799

[2000] training's auc: 0.91527 valid\_1's auc: 0.879359

[3000] training's auc: 0.927675 valid\_1's auc: 0.887254

[4000] training's auc: 0.935435 valid\_1's auc: 0.890105

[5000] training's auc: 0.941904 valid\_1's auc: 0.891646

[6000] training's auc: 0.947745 valid\_1's auc: 0.892288

[7000] training's auc: 0.953197 valid\_1's auc: 0.892084

[8000] training's auc: 0.95833 valid\_1's auc: 0.892595

[9000] training's auc: 0.963095 valid\_1's auc: 0.892093

[10000] training's auc: 0.967417 valid\_1's auc: 0.891904

Early stopping, best iteration is:

[7730] training's auc: 0.956929 valid\_1's auc: 0.892635

Fold 3

[59935 59936 59937 ... 80011 80013 80014]

[ 0 1 2 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.886877 valid\_1's auc: 0.865543

[2000] training's auc: 0.914624 valid\_1's auc: 0.887235

[3000] training's auc: 0.927242 valid\_1's auc: 0.89372

[4000] training's auc: 0.935303 valid\_1's auc: 0.896423

[5000] training's auc: 0.941748 valid\_1's auc: 0.897352

[6000] training's auc: 0.947511 valid\_1's auc: 0.897486

[7000] training's auc: 0.952921 valid\_1's auc: 0.89716

[8000] training's auc: 0.958047 valid\_1's auc: 0.897348

[9000] training's auc: 0.962791 valid\_1's auc: 0.897304

Early stopping, best iteration is:

[6124] training's auc: 0.948196 valid\_1's auc: 0.897632

Fold 4

[ 79935 79951 79958 ... 100411 100421 100423]

[ 0 1 2 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.886477 valid\_1's auc: 0.863409

[2000] training's auc: 0.91464 valid\_1's auc: 0.886288

[3000] training's auc: 0.927108 valid\_1's auc: 0.893696

[4000] training's auc: 0.935091 valid\_1's auc: 0.896725

[5000] training's auc: 0.94143 valid\_1's auc: 0.897585

[6000] training's auc: 0.947264 valid\_1's auc: 0.898335

[7000] training's auc: 0.952702 valid\_1's auc: 0.898173

[8000] training's auc: 0.957738 valid\_1's auc: 0.898341

[9000] training's auc: 0.962445 valid\_1's auc: 0.898251

Early stopping, best iteration is:

[6093] training's auc: 0.947794 valid\_1's auc: 0.898434

Fold 5

[ 99965 99966 99967 ... 120323 120328 120354]

[ 0 1 2 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.886379 valid\_1's auc: 0.866762

[2000] training's auc: 0.914599 valid\_1's auc: 0.889633

[3000] training's auc: 0.926978 valid\_1's auc: 0.896791

[4000] training's auc: 0.934983 valid\_1's auc: 0.899414

[5000] training's auc: 0.941377 valid\_1's auc: 0.90074

[6000] training's auc: 0.947253 valid\_1's auc: 0.9011

[7000] training's auc: 0.952652 valid\_1's auc: 0.900766

[8000] training's auc: 0.95777 valid\_1's auc: 0.900873

Early stopping, best iteration is:

[5905] training's auc: 0.946757 valid\_1's auc: 0.901241

Fold 6

[119960 119961 119962 ... 140649 140657 140661]

[ 0 1 2 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.88672 valid\_1's auc: 0.866219

[2000] training's auc: 0.914835 valid\_1's auc: 0.887328

[3000] training's auc: 0.927183 valid\_1's auc: 0.893592

[4000] training's auc: 0.935352 valid\_1's auc: 0.896797

[5000] training's auc: 0.941866 valid\_1's auc: 0.897569

[6000] training's auc: 0.947682 valid\_1's auc: 0.898004

[7000] training's auc: 0.953129 valid\_1's auc: 0.898095

[8000] training's auc: 0.958095 valid\_1's auc: 0.89806

[9000] training's auc: 0.962689 valid\_1's auc: 0.897824

[10000] training's auc: 0.967073 valid\_1's auc: 0.897745

[11000] training's auc: 0.971011 valid\_1's auc: 0.897913

Early stopping, best iteration is:

[8189] training's auc: 0.958971 valid\_1's auc: 0.898251

Fold 7

[139923 139924 139925 ... 160291 160304 160366]

[ 0 1 2 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.88654 valid\_1's auc: 0.863204

[2000] training's auc: 0.914499 valid\_1's auc: 0.884992

[3000] training's auc: 0.926917 valid\_1's auc: 0.892451

[4000] training's auc: 0.93486 valid\_1's auc: 0.895764

[5000] training's auc: 0.941345 valid\_1's auc: 0.897195

[6000] training's auc: 0.947219 valid\_1's auc: 0.897623

[7000] training's auc: 0.952565 valid\_1's auc: 0.897981

[8000] training's auc: 0.957695 valid\_1's auc: 0.897761

[9000] training's auc: 0.962541 valid\_1's auc: 0.897846

Early stopping, best iteration is:

[6875] training's auc: 0.951909 valid\_1's auc: 0.898155

Fold 8

[159971 159972 159973 ... 180458 180464 180481]

[ 0 1 2 ... 199997 199998 199999]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.887104 valid\_1's auc: 0.871879

[2000] training's auc: 0.914433 valid\_1's auc: 0.892116

[3000] training's auc: 0.926872 valid\_1's auc: 0.899571

[4000] training's auc: 0.934969 valid\_1's auc: 0.902594

[5000] training's auc: 0.941291 valid\_1's auc: 0.903082

[6000] training's auc: 0.947041 valid\_1's auc: 0.903124

[7000] training's auc: 0.952565 valid\_1's auc: 0.903

[8000] training's auc: 0.957774 valid\_1's auc: 0.90268

Early stopping, best iteration is:

[5525] training's auc: 0.944361 valid\_1's auc: 0.903318

Fold 9

[179945 179948 179950 ... 199997 199998 199999]

[ 0 1 2 ... 180458 180464 180481]

Training until validation scores don't improve for 3000 rounds.

[1000] training's auc: 0.886359 valid\_1's auc: 0.866674

[2000] training's auc: 0.914671 valid\_1's auc: 0.887478

[3000] training's auc: 0.927293 valid\_1's auc: 0.894258

[4000] training's auc: 0.935359 valid\_1's auc: 0.897028

[5000] training's auc: 0.941739 valid\_1's auc: 0.898116

[6000] training's auc: 0.947616 valid\_1's auc: 0.898924

[7000] training's auc: 0.953067 valid\_1's auc: 0.898635

[8000] training's auc: 0.958137 valid\_1's auc: 0.898756

[9000] training's auc: 0.963015 valid\_1's auc: 0.898549

[10000] training's auc: 0.967389 valid\_1's auc: 0.898225

Early stopping, best iteration is:

[7855] training's auc: 0.957462 valid\_1's auc: 0.898989

CV score: 0.89822

In [134

##### feature importance in python

cols = (feature\_importance\_df[["Feature", "importance"]]

.groupby("Feature")

.mean()

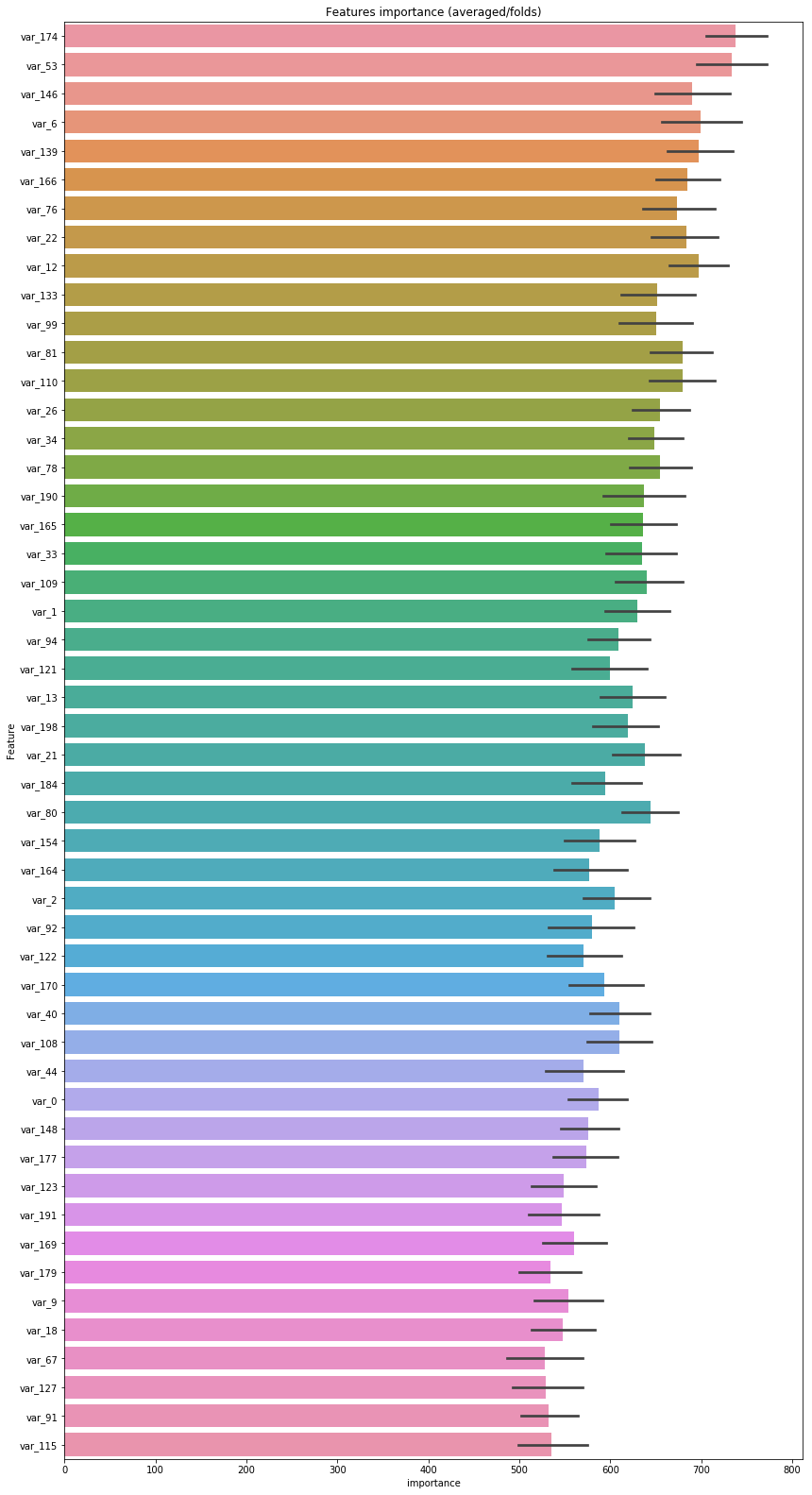
.sort\_values(by="importance", ascending=False)[:50].index)

best\_features = feature\_importance\_df.loc[feature\_importance\_df.Feature.isin(cols)]

plt.figure(figsize=(14,28))

sns.barplot(x="importance", y="Feature", data=best\_features.sort\_values(by="importance",ascending=False))

plt.title('Features importance (avg/folds)')



IN R(i cound't able to install light GBM So I did not get any output but the code is very similar to python)

x<-c('pscl', 'ROCR', 'lightgbm', 'methods', 'Matrix')

install.packages(x)

lapply(x, require, character.only = TRUE)

#creating lgb dataset

trainm = sparse.model.matrix(target ~., data = train\_file)

train\_label = train\_file[,1]

valm = sparse.model.matrix(target~., data= val\_file)

val\_label = val\_file[,"target"]

train\_matrix = lgb.Dataset(data = as.matrix(trainm), label = train\_label)

val\_matrix = lgb.Dataset(data = as.matrix(valm), label = val\_label)

#install.packages(file.path("C:/Users/HP/Documents/R/win-library/3.5", "LightGBM", "R-package"), repos = NULL, type = "source")

param = list(

bagging\_freq= 5,

bagging\_fraction=0.4,

boost\_from\_average=false,

boost= 'gbdt',

feature\_fraction=0.05,

learning\_rate= 0.01,

max\_depth= -1,

metric='auc',

min\_data\_in\_leaf= 80,

min\_sum\_hessian\_in\_leaf= 10.0,

num\_leaves= 13,

num\_threads= 8,

tree\_learner= 'serial',

objective= 'binary',

verbosity= 1

)

valid = list(test = val\_matrix)

bst = lightgbm(params = params, train\_matrix, valid, nrounds=1000000,

early\_stopping\_rounds = 3000,

eval\_freq=1000,

seed=44000)

p = predict(bst, valm)

predicted\_lgm = ifelse(p > 0.5,1,0)

confusionMatrix(factor(predicted\_lgm), factor(val\_file$target))

## Final prediction

#### In Python

Using top 2 model for prediction

new\_test= test.drop(['ID\_code'], axis=1)

standardized\_test = preprocessing.scale(new\_test)

NB\_Predictions = NB\_model.predict(standardized\_test)

lgb\_prediction=clf.predict(new\_test)

test['target1']= NB\_Predictions

test['target2']=lgb\_prediction

#### IN R

test$target1<-predict(bst,test[,-1])

scale\_test=as.data.frame(scale(train[,2:201]))

test$target2<-predict(NB\_model,scale\_test,type='class')

Summary:

for this project

1: all two hundered features are independent of each other So there is no chance of feature reduction

2:there is class inbalance problem

3:decision tree and random forest are not good model for this classification

4:Naive bayes works pretty well campared to other basic model

5: I have used light GBM as a black box model which is providing a much better result compared to other model.